PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 17/30 — A2

(11) International Publication Number: WO 97/40457

(43) International Publication Date: 30 October 1997 (30.10.97)

(21) International Application Number: PCT/US97/06468

(22) International Filing Date: 17 April 1997 (17.04.97)

(30) Priority Data:

60/016,368	19 April 1996 (19.04.96)	US
60/017,483	26 April 1996 (26.04.96)	US
60/024,571	26 August 1996 (26.08.96)	US

(71) Applicant: INTERGRAPH CORPORATION [US/US]; One Madison Industrial Park, Huntsville, AL 35894-0001 (US).

(72) Inventors: BISBEE, Robert, T.; 217 Evalyn Street, Madison, AL 35738 (US). EVANS, Cynthia, H.; 1743 Cedar Hurst Road, Pulaski, TN 38478 (US). THOMAS, Robert, O.; Apartment 1624, 3784 University Drive, Huntsville, AL 35816 (US).

(74) Agents: SUNSTEIN, Bruce, D. et al.; Bromberg & Sunstein L.L.P., 125 Summer Street, Boston, MA 02110-1618 (US).

(81) Designated States: European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

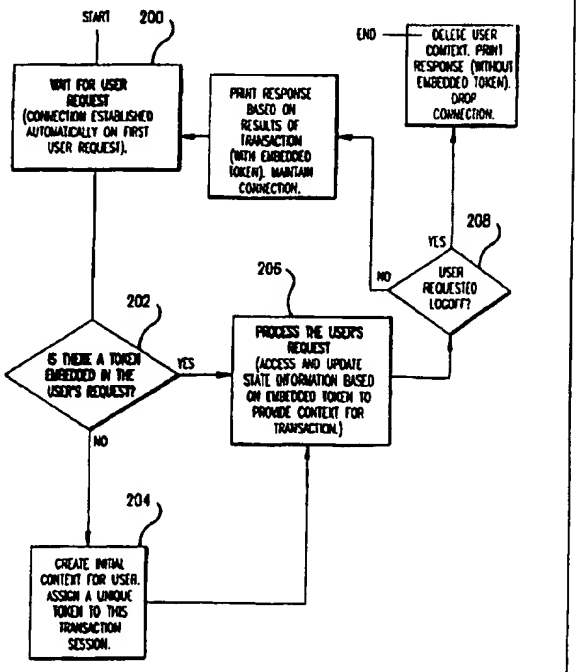
Published

Without international search report and to be republished upon receipt of that report.

(54) Title: SYSTEM AND METHOD FOR DATA ACCESS

(57) Abstract

A client-server system utilizing a stateless protocol to access server data over a network (55). Such a system may implement online transaction processing, where a client (60) utilizes specialized computer software to provide entry forms to indicate information to be retrieved from an application server (50). A unique identifier is generated to identify a particular client, and is embedded into communications between the client (60) and the application server (50), the presence of the identifier causing the application server (50) to maintain an active network connection (55) to the client, as well as allowing the application server to maintain server-side state information, such as information to be distributed to the client computer, the client identifier, and other desired information regarding the client's activities. Maintaining an active connection, as opposed to continually dropping and reinitiating contact, reduces network overhead. A database server (40) may operate in conjunction with the application server (50) through communications over a second network connection (45) for storing the server-side information. The application server (50) may be implemented using Web server technology. If the HTTP Hypertext Transport Protocol is used on a network (55) to communicate with the client (60), then the Client (60) may use any operating system supporting the HTTP protocol.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav Republic of Macedonia	TM	Turkmenistan
BF	Burkina Faso	GR	Greece			TR	Turkey
BG	Bulgaria	HU	Hungary	ML	Mali	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MN	Mongolia	UA	Ukraine
BR	Brazil	IL	Israel	MR	Mauritania	UG	Uganda
BY	Belarus	IS	Iceland	MW	Malawi	US	United States of America
CA	Canada	IT	Italy	MX	Mexico	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NE	Niger	VN	Viet Nam
CG	Congo	KE	Kenya	NL	Netherlands	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NO	Norway	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	NZ	New Zealand		
CM	Cameroon			PL	Poland		
CN	China	KR	Republic of Korea	PT	Portugal		
CU	Cuba	KZ	Kazakstan	RO	Romania		
CZ	Czech Republic	LC	Saint Lucia	RU	Russian Federation		
DE	Germany	LI	Liechtenstein	SD	Sudan		
DK	Denmark	LK	Sri Lanka	SE	Sweden		
EE	Estonia	LR	Liberia	SG	Singapore		

SYSTEM AND METHOD FOR DATA ACCESS

DESCRIPTION

TECHNICAL FIELD

This invention relates to computer systems, and more particularly to client-server computer systems that provide access to on-line data.

BACKGROUND OF THE INVENTION

A common use of client-server computer systems is the retrieval of information by the client system of data stored on a server system over a network connection. The network involved in such a retrieval may be the public Internet or a private Intranet. An Intranet is a private network, generally augmenting or replacing a Local Area Network (LAN) within an office, yet based upon the widely available Internet technology. One part of the Internet technology of current interest is the HTTP Hypertext Transaction Protocol, the protocol that carries the network communications between many existing client and server systems. As with most networking protocols, HTTP is a high-level set of networking functions that rides on top of lower-level basic network protocols. A good example of a superset protocol is the well-known TCP/IP high-level protocol, which is built on top of the low-level IP protocol. IP is essentially the assembly-language of networking protocols. It offers very basic low-level functions for forming a network connection between two locations on the network (such as between a client and a server), but IP offers very little error handling recovery. In contrast, TCP/IP builds off of IP to provide features to guarantee error-free transmissions over the network. In this same fashion, the HTTP protocol builds on present day networking protocols to allow efficient client-server connections over a network where it is expected that a large volume of data (principally due to graphics information) will be transferred between the two.

In its most common form of use, HTTP is a stateless protocol. The "state" of a client-server transaction refers to what the client is doing/requesting now, in relation to the past

history of requests to the server. A stateless protocol means that the communications are one-way requests, and that the requests are not being tracked. Thus, in the client-server context, every time a client wanted information, the client would have to make fully encapsulated requests to a server. The server would receive the request, process it and send the client the result, and then immediately drop the connection between the server and client. After the drop there is no persisting record at the server of information related to the communication channel that was just closed, and no information related to the client system or to the specific user of the client system. Therefore, if the client needed the "address" information again from a previous request, it could not simply transmit a "send me that address-data again" request as the reference to "address-data" requires the server to have maintained a state history for the client in order for the server to know what was sent before.

Consequently, follow-on communications between the client and server that are logically connected, for example follow-on communications regarding the same previous user or the same previous topic, require that a new communication channel be established for the follow-on communications without reference to or use of prior information. This process requires a repeat expenditure of system overhead by both the client and the server systems to set up a new channel for each HTTP communication, as well as expenditure of other resources to re-transmit information from the client to the server that may be needed to access the information being sought. In effect, the client is required to transmit state information to the server if the client wishes to perform an ongoing dialog with the server.

For many practical applications, the stateless connection is adequate. For example, when a human user is utilizing a client system to access a data source located remotely using the Internet, related follow-on requests are relatively infrequent, so dropping and re-establishing a new communication channel is efficient, and client system or user information is often not required for data access. In other more data intensive applications, however, client information and/or communication channel information is frequently important to retain. Such applications include on-line transaction processing system (OLTP), which includes both reading and changing data, or on-line access processing (OLAP), which includes only reading data.

In such data intensive systems, requiring the client to transfer state information along with the data requests can make the client-server relationship extremely inefficient. Having the server retain client state information over a stateless connection is a way to avoid the inefficiency as well as to increase total system capacity; these two benefits are features of the present invention.

For present purposes, the invention will be described here in terms of an OLTP, however it will be understood that the present invention has broader application than the OLTP system described. The invention may find utility in any communications system requiring multiple interactions or transactions within a user session, and which provides client context within the server for follow-on interactions.

An on-line transaction processing system (OLTP) typically provides a computer system comprising a client-server arrangement having three tiers, although a two tier system will also be described below for completeness. A plurality of client systems at the first tier level having keyboards and display monitors provide input and output directly to human users who have a need to remotely access a database of information records. Network connections connect the client systems to a second tier comprising application servers that provide the application software such as an inventory control system or an airline reservation system that deals with user requests, performing data collection and validation and supplying user responses and other application services. Network connections connect the application servers to a third tier comprising, typically, a single or small number of database servers that contain the basic information records, e.g., inventory or airline seats, that are queried and updated as the human users interact with the data through the application system. Networks in an OLTP may be local area networks or wide area networks as the physical distances between the client and the server systems requires.

In the past, most OLTP systems have used an operating system sold under the UNIX brand name by a number of commercial suppliers, in the computer systems at all three levels, client system, application server and database server, in order to control software operation and network communications with the other tiers. More recently, it has been desired to use an operating system sold under the Windows brand name as Windows NT Server available from

Microsoft Corporation of Redmond, Washington at the application server tier, and, in some cases, one or more different versions of Windows at all three tiers.

Using Windows NT Server as the application server permits the application server to be advantageously implemented using a combination of current commercially available hardware and software products, the combination generally referred to as a Web server, with most development implemented under the Windows operating system. A Web server comprises a hardware and software combination suitable for operation as a server on the World Wide Web, an increasingly commercially-oriented segment of the publicly available Internet network. When the Windows operating system is used in the application tier, however, experience shows that the overall capacity of the OLTP system is substantially less for the same hardware than the capacity when the UNIX operating system is used. Solving this capacity problem would encourage OLTP systems to be implemented using Web server technology on a private Intranet network, or using Web servers as OLTP application servers or data servers on the World Wide Web itself.

Additional details on the background and the problem being solved by the present invention will be found in the article "Putting the Data Warehouse on the Intranet", appearing in *Internet Systems*, May 1996, a periodical published by Miller Freeman as a supplement to the periodical DBMS.

SUMMARY OF THE INVENTION

The present invention provides an increase in capacity in a client-server system that uses a stateless protocol to access server data. The invention does this by improving network communication efficiency over the network linking the server system to the client system. For each initial client system transaction request, the server generates a unique identifier, or token, which is returned over the network to the client system, the receipt of which token acts to hold the logical connection to the server in an active or open condition until the client system is finished with the connection and releases it. At the server, the token is used to identify client state information, also referred to as user context information, retained at the server for use with follow-on communications with the client. Follow-on communications

from the client include the token, which the server uses to identify the stored client state information.

The present invention further reduces network overhead by keeping alive the network connection between server system and client system, thereby reducing total system overhead. As a result, in one embodiment, the server creates only a single or a small number of operating system processes within the server system in response to a comparatively large number of client system transaction requests. Used individually or in combination, these improvements increase the capacity of a client-server system based upon a stateless communication protocol.

In one preferred embodiment of the present invention, the data access system would include a client computer and a server. At least one database maintained on the server, the database having information to be distributed to the client computer and a token associated with the client, wherein the token represents the status of communications between the client computer and the server. The status of communications includes the identity of the client and the context of any communications between the client computer and the server. There would also be a communication path between the client computer and the server, each client computer programmed with a specialized computer software program providing an at least one data-entry field to allow a particular user of the client computer to complete a request of information to be retrieved from the server. There would also be a communication path between the server and the client, allowing the server to communicate a response to the particular user's request. In preferred embodiments, the communications between a client and a server may be encrypted, using a proprietary or public-key cryptosystem, allowing for secure communication over a communication link (e.g. network, satellite broadcasts, or cable TV) lacking in security.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 shows a block diagram of a prior art three-tier OLTP system based on the UNIX operating system.

FIG. 2 shows a block diagram of an embodiment of a three-tier OLTP system according to the present invention using the Windows NT Server operating system at the application server tier.

FIG. 3 shows a block diagram of an embodiment of a three-tier OLTP system implemented as a two-tier system wherein the database server software and the application server software for a particular database are provided within a single computer system using the Windows NT Server operating system.

FIG. 4 shows diagrammatically certain memory and software interaction between application system software and client system software to effect improvements of the present invention.

FIG. 5 is a flow-chart of the interaction between a client and a server communicating in accordance with an embodiment of the present invention.

FIG. 6 is a more detailed view of two steps contained within the FIG. 5 flow-chart.

FIG. 7 is a flow-chart showing the general handling of cookies by a preferred embodiment.

DESCRIPTION OF SPECIFIC EMBODIMENTS

FIG. 1 is an example prior art three-tier OLTP implementing an inventory control system. In FIG. 1, database server **10** provides common access to information records for users connected to on line client system **30**. Server **10** includes magnetic disk or other non-volatile storage that contains a database of information records necessary for implementation of an inventory control system. Server **10** manages the database using one of a number of commercially available database management systems, such as the Oracle system sold by Oracle Corporation of Redwood Shores, California.

Database server **10** receives and responds to requests for database services over network **15** from one or more application servers **20** which implement a data-dependent software application, such as an inventory control system. Application server **20** receives and responds to requests for inventory control services over network **25** from one or more client systems **30**. One or more human users interact with each client system **30** using keyboard,

mouse, display monitor, printer or any number of other input/output devices to utilize the inventory control system.

As shown in FIG. 1, prior art OLTP systems of this type generally utilize the UNIX operating system at each tier of the system. Database server **10** and application server **20** typically communicate over network **15** utilizing calls for database services that are native to the particular database manager being utilized, Oracle in this specific embodiment, or else the ODBC Open Database Connectivity protocol which is an industry standard database communication protocol. All commercially important database management systems generally available, including Oracle, will accept standard ODBC requests and will provide standard ODBC responses over network **15.** In this document ODBC is used illustratively throughout, but it will be understood to represent generally ODBC, the transaction language native to the specific database management system (DBMS), or any other transaction language suitable for database communications.

Application server **20** and client system **30** using the UNIX operating system typically communicate over network **25** utilizing the Telnet UNIX industry standard communication protocol. The UNIX operating system at application server **20** can handle a significant number of simultaneous Telnet connections, so that total OLTP system capacity is sufficient for its intended purpose.

FIG. 2 shows a block diagram of an embodiment of a three-tier OLTP system according to the present invention using the Windows NT Server operating system at the application server tier. Database server **40** and application server **50** communicate over network connection **45** using the ODBC or other protocol in a manner similar to the prior art system. Application server **50** is advantageously implemented using Web server technology under Windows NT Server. As part of this improvement, the HTTP Hypertext Transport Protocol is used on network **55** to communicate with client system **60**. Client system **60** may use any operating system supporting the HTTP protocol, such as UNIX or one of the Windows products.

In the embodiment of FIG. 2, the Web server features of the application server **50** are implemented in software using the Visual C++ Version 4.1 suite of development software

from Microsoft. Any programming language could be used, but this suite of software is particularly advantageous. This suite includes the Microsoft Foundation Classes and the ISAPI Internet Server Application Program Interface software. The ISAPI provides rudimentary Web server and HTTP features that can be built upon using the Visual C++ compiler provided in the suite. Similarly, in client system **60** communication with application server **50** over network connection **55** using the HTTP protocol is implemented using the ISAPI expanded in an appropriate fashion to cooperate with the software implemented at server **50**. In this way, application server **50** and client system **60** cooperate to reduce the communication load over network connection **55.**

FIG. 3 shows a block diagram of an embodiment of a three-tier OLTP system implemented as a two-tier system wherein the database server software and the application server software for one particular database of the OLTP system are provided within a combination server **70** using the Windows NT Server operating system. In the embodiment of FIG. 3, ODBC or other protocol communication between the database management system and the application system software is accomplished with direct software communication between the two software systems rather than using a network connection. The Windows NT Server operating system cooperates by supporting ODBC and other communication between separate systems.

Other particular databases of the OLTP system of FIG. 3 may be implemented with separate servers for the database software and for the application software, as previously shown in FIG. 2. Client system **80** and network connection **75** operate in similar fashion to the system of FIG. 2. Improvements in OLTP system capacity of the present invention may be applied to advantage in the case of a combination server arrangement shown in FIG. 3.

FIG. 4 shows diagrammatically certain memory and software interaction between application system software and client system software to effect improvements of the present invention. When a new initial client server request is received by the application server **50**, the application reserves an area in volatile memory for the opened communication channel, and creates a token, a unique element of data, to identify the new user session and the new client. This token is sent back to the client to identify the new session, which the client retains

in volatile memory. Subsequent communications between client and application include the token to identify the communication session being utilized, and cause the user or client context information to be retained on the server as long as needed. So long as the context information is retained, the application server is able to make subsequent and repeated interchanges with the database server in response to client requests during the session.

The client has responsibility for notifying the application when the session is ended and is to be dropped. In response to a drop notification, the application releases the memory that was reserved for the context information and the token is released.

FIG. 4 also illustrates a portion of volatile memory used in the token and context arrangement just described. Application server **50** reserves a memory area **51** when a new initial client communication is received, and a token is created and stored away as a pointer to the client state information stored in memory area **51**. Various items of client state, or context, information may be advantageously stored in memory area **51**, depending on the needs of the specific application. In the present illustrative embodiment representing an inventory control system, an identifier of the current warehouse of interest to the user of the client system may be stored in memory area **51**. The warehouse identifier information can then be included in ODBC or other protocol interactions with the DBMS. The current warehouse identifier may be changed from time-to-time as the user's utilization of the inventory system continues over subsequent inquiries during the same session, and the token returned from the client with the subsequent inquiries provides an effective means for locating and updating the user context information.

The token is transmitted over network connection **55** to client system **60** where it is retained in memory area **61**, which may be volatile or non-volatile, depending on the embodiment, until no longer needed. In some embodiments, the token may be retained in volatile memory only until the next transmission of data back to the application server wherein the token is included in the return data. In other embodiments, the token may be retained at the client system for a longer period of time, depending on the advantageous use that can be made of the token by the client system, as, for example, until the end of the session. The token may be retained at the client system at least until the text transmission of

data back to the application server, and for a longer period. for example, in embodiments wherein client system **60** supports a plurality of human users. and user context information useful to the client is maintained in the client system, or embodiments wherein the token serves different or additional functions, such as a role in security features or other functions related to database functions, application functions, client functions or communication functions.

In any event, client system **60** transmits the token to application system **50** in one or more subsequent communications to identify the session and the memory area **51** allocated to the client state information. While the session with the client is open, application server **50** is able to make any number of database interchanges with the database server over the ODBC communication channel, either a channel provided by network as illustrated in FIG. 2, or a channel provided by software as illustrated in FIG. 3.

Additional improvements in data access system capacity can be obtained by using the token and client context arrangement described above in combination with a mechanism for maintaining a continuous network connection between the client system and server system. An advantageous means for accomplishing this result is provided by the keep_alive feature provided by certain implementations of the HTTP protocol. Such an implementation is found in the HTTP protocol provided by the Web server software available from Microsoft for use with Windows NT Server, and by the Microsoft Internet Explorer Web browser software that operates in conjunction with Windows NT. Using this browser or equivalent software in the client system of FIG. 2 or FIG. 3 makes the keep_alive feature available. When a keep_alive request is issued by one system to the other over the network connection, the recipient of the request maintains the communication channel until it is released by the requester. In this way system overhead is further reduced and system capacity is further increased for multiple interactions or transactions within a session. In the present invention, the HTTP communication channel between the client and application servers can be made continuous as long as reasonably needed.

In the prior art, it is known for an application server to initiate a keep_alive request to the client system when the application server has a need to transmit relatively large amounts

of data requested by the client, such as graphics data, usually in repeated transmissions. By making a keep_alive request to the client, the application prevents the communication channel from being dropped until the application has finished transmitting all the data requested during subsequent transmissions by the application.

In the present invention, the client makes a keep_alive request to the application in anticipation of a number of repeated interchanges of information with the application. Having received a keep_alive request from the client, the Web server, if designed to operate with the keep_alive option, can choose to honor the keep_alive request or not. In the embodiments of the present invention wherein keep_alive is used in conjunction with the token and client context arrangement previously described, the Web server, in conjunction with the application software, honors the keep_alive request and returns a notification to that effect to the client. Thereafter, communications over the HTTP network connection is more efficient since the connection is not taken down after each message and re-established anew with the following message. This is a particularly advantageous arrangement that enhances the efficiency of the token and client context arrangement previously described.

FIGS. 5 and 6 show, in flow-chart form, an over-view of the interaction between a client system and the server system in an illustrative application server computer program for a simplified example inventory control system having features in accordance with the present invention that has been described hereinabove. In FIG. 5, step **100**, a client initially types in the Universal Resource Locator (URL) address of the server system to contact. The server, at step **102**, verifies whether or not the server already has context data for the contacting Client. If there is no context data for the client, at step **104**, the server returns a login form to the client, which requests that the client submit identification information. At step **106**, the client submits the filled in login form, and the server, at step **108**, creates user context information and the cookie associated with that context information. Then, at step **110**, the server sends the client a menu of actions/functions the client can request the server perform, where embedded within the form is the cookie associated with the client's context information. At step **112**, the client selects an action from the menu, and this selection is transmitted to the server. Embedded within this and other communications with the server is the cookie value

generated in step **108**. If the client has not chosen to quit the program at step **114**, resulting in an end to the connection with the server at step **116**, the program flow loops back to step **102** in which the server verifies there is a context for the client (which has just been created at step **108**), and then at step **118** the server evaluates the selected action/function chosen in step **112**. If the request is not valid, either due to the current context for the user, garbled transmission, or some other error, at step **122**, the server sends the client an error message.

At step **124**, of FIGS. 5 and 6, the server performs the chosen action/function. FIG. 6 shows at steps **150-162**, for one preferred embodiment of the present invention, the available actions/functions the client may request. In a preferred embodiment, at step **150** the server evaluates whether the client chose to print new order results, step **152**, print delivery results at step **154**, print payment results at step **156**, print stock levels at step **158**, print order status information at step **160**, or process a login form at step **162**. In preferred embodiments, the client login form is treated as any other request the client may make.

At step **126**, of FIGS. 5 and 6, the server transmits to the client the results of performing the requested function. FIG. 5 indicates that after returning results in step **126**, program control loops back to step **110** in which the user is then presented with the menu of actions/functions that may be performed.

Described hereinbelow is a more detailed-view of the illustrative inventory control program utilized in FIG. 5 and FIG. 6. The illustrative program was compiled with the Microsoft Visual C++ Compiler Version 4.1 to form an embodiment of the present invention, but another compiler and or language may have been used instead. Also note that the following description is not intended to be a fully functional nor fully operational inventory control application.

In operation, the inventory control application provides HTML Hypertext Markup Language forms to the client system through Web server software. The user of the client system, using an available Web browser such as the Microsoft Internet Explorer Web browser, types information into the forms and selects screen buttons appearing on the forms. The Web browser at the client system returns each filled-in form to the application system, which parses the form (Fig. 5 step **118**) and selects the information of interest (step **124**). In

response to a login form (step **106**) identifying a new user session, the application will reserve an area of user context memory and assign a token (step **108**), referred to as the "cookie" in the computer program code described hereinbelow. The cookie is inserted into each of the forms that will thereafter be sent to the user of the client system. When the received form is filled in by the user and returned to the application for processing, the cookie that was placed in the form by the application identifies the user session for the application. The application uses the cookie to locate the user context information to aid further processing of the information provided on the form, or for communicating with the database server, or both. When the user presses the "Exit" screen button on the form presented by the client system (step **114**), the form data is returned to the application which determines that the Exit button was pressed. The application then releases the user context memory and releases the cookie (token) for re-use by another user, ending the current user session (step **116**).

In a preferred embodiment of the inventory control program, basic utility functions are needed to interface a Web browser with a database storage system. Common to most support functions, is that each contains a reference to a cookie (token), and this cookie is used to load a context for a particular user. The context is a memory area reserved for the cookie in each form that will be returned to the user. For any given transaction by a client being processed by a server, a preferred embodiment will utilize data structures containing fields for tracking data relevant to the transaction taking place between the client and server. For example, once such structure, referenced herein as the raw data form structure, contains entries for identifying the form in use by the client, entries for a product, service or resource being reviewed by the client, quantity indicators, and other data concerning a transaction taking place with a client. Such data structures are then used by the support functions to enter and manipulate transaction data.

One such support function is PrintStaticPage(), which is used to send responses back to a client's browser. As described hereinabove, generally a client user will connect to a server computer through a Web browser, and be presented with a form allowing the user to indicate an action to be performed by the server, such as to perform a price check. In response to this request, different components to the invention perform the database retrieval operation,

and the PrintStaticPage() is then called to dynamically create a formatted HTML page containing the results of the request. This formatted HTML page is then sent back to the client browser so that the results of the request may be seen.

Another support function is VerifyNewOrderLine(), which verifies that a user's input to a form is valid, and returns an error indicating the location of any errors (such as unexpected blank responses). This function is representative of a set of functions which each verify a certain type of input field. VerifyInt(), VerifyShort(), etc. are also in this set. VerifyNewOrderLine is more complex in that it checks an entire order line (composed of multiple fields) for validity. It is called during a New Order transaction, and would also be called from PrintNewOrderResults() (which is also a function representative of a set of functions, one for each specific transaction), and which interprets a user's response to input forms and generates the appropriate response page data (which is then given to PrintStaticPage() for transmittal to the client browser). This function is passed a pointer to the user's query data, and a pointer to a raw form data structure containing all possible fields that may be assigned values during interacting with a particular client. Initially this function (as do most functions in a preferred embodiment) uses the user's assigned cookie to retrieve the current context data for that client from the server. When the client submits a request form for processing, all entries upon the form are passed to this function as a long character string, and this string is parsed and stored into the raw data form structure. After parsing, the client's context information on the server is updated to reflect the new client data, and subsequent references to the user's cookie will include the newly tracked data.

In a preferred embodiment, for communicating with a database, entries from the raw form are copied into a predefined database-access field. This access field, as well as other database related structures and operations used by a preferred embodiment to interact with a SQL-type database, have been defined in the Microsoft TPC-C Benchmark Kit, version 2.04 (hereinafter Benchmark Kit). Once appropriate values have been copied into the access field this field is then passed to the SQLNewOrder() function (defined by the Benchmark Kit) to effect a database query.

Other support functions, such as PrintPaymentResults(), PrintOrderStatusResults(), PrintDeliveryResults(), PrintStockLevelResults(), are also passed as arguments to the function, a pointer to the user's query data, and a pointer to the raw form data for storing the parsed query data. This parsed data is then used to fill in data structures relevant to the particular action to be performed. Thus, for example, a predefined structure of type PAYMENT_DATA is filled with data concerning exactly when in time the transaction took place, the nature and status of the transaction, etc. This retrieval is accomplished by passing the predefined structure to a SQLOrderStatus() function (defined by the Benchmark Kit), which queries the appropriate SQL database and fills in the predefined structure.

In addition to the support functions, in a preferred embodiment of the inventory control program, are handling functions which interpret user responses to the forms and provide some of the basic functions found in an inventory control system. In each user response form, the cookie is retrieved from the form, converted from a string into a number, and the number used to index into an array (m_pContext) of user context information from an operating server.

One such handling function is ProcessLoginForm(); this function generates a request for the allocation of a new user context memory area and the creation of an associated cookie that refers to it. Another handling function, ParseFormData(), breaks up the data string comprising the form sent from the client system into individual data units, the cookie being one such data unit. Another handling function, SelectPage(), takes the cookie returned from the client system in string form, converts it into a number that can be used as an index into the user context array, and checks its validity. Then the function branches into one of a number of paths that process a specific request made by the user (step **124**), and determines which output page should be displayed back to the user. In a preferred embodiment of the inventory control program example, each of the forms labeled nform and oform are created at the outset during login, and initialized with the cookie for future use. Input received from a user are tagged, such that if the user pressed the exit button, the server program would receive an action case of 'E', causing a function call to release the user context memory area and the cookie. Other actions are similarly tagged for processing by the server.

Another handling function is HttpExtensionProc(); this function, in response to a keep_alive request from the client, this function complies with the request and returns a value to the client that keeps the network connection active. In a preferred embodiment, each client communication from the Web browser to the Web server includes a request to keep the connection alive. The Web server complies and returns a similar request to the client. In this way the network connection is retained so long as needed. This in contrast to the normal browser connection mode in which all actions are atomic, and a new network connection made to the server each time the client attempts to take any action. For communications transmitted with keep_alive in this specific embodiment, an additional data item specifying the length of the transmitted message is appended when a message is sent across the network connection. In this way, the receiver can determine when the transmitted message has been completely received. After the application software processes an Exit button press, as previously described hereinabove, the response message to the client does not return a keep_alive request, permitting the connection to be dropped.

Another handling function is CreateContext(); this function (step **108**) reserves a cookie for the user session by finding a free slot in an array of pointers to context memory entries, and the user context memory area associated with the cookie. The contexts are stored upon the server, and each user context contains working copies of the response forms with the cookie entered into them. When a context is created, a connection is made to the database and information regarding this connection is stored within the context. In the inventory control program, this context also contains information such as warehouse and district identifiers provided by the user.

Related to CreateContext() is the FreeContext() function that deletes the user context and marks the array slot free for future use. In a preferred embodiment, this function receives as an argument a cookie, and if a user session is associated with the cookie, the associated database connection is closed, and associated resources released.

FIG. 7 is a flow-chart showing the general handling of cookies by a preferred embodiment. Initially, the program waits **200** for a user request to be received; during the first request, a connection is automatically established with a server. A check is made **202** to

determine whether a token was embedded in the request, and if not, an initial context is created **204** on the server. After the context has been created, the users' request is then processed **206** according to the request type and associated data. During processing, the client's sate information is updated according to the transaction processed. After processing, a test is made **208** to determine if the action taken by the user was to perform a logoff from the server. If so, then the context for the user is deleted **210** and a logoff response (without a cookie/token) is presented to the user and the user drops the connection. If it was not a logoff request, then the user is sent a response **210** (containing a cookie) corresponding to the results of the requested transaction. The connection is not dropped by the user, and the server continues to wait **200** for further requests from the user.

In addition to the embodiments disclosed above, additional embodiments of the invention will be apparent to those with ordinary skill in the art without departing from the teachings of the invention. For example, instead of basing the communications between the client system and the server system solely upon HTML or HTTP, Java (a platform-independent multi-threaded, dynamic general-purpose programming environment for creating applets and applications for distributed networks by Sun Microsystems Inc.) or its equivalent may be used to augment the functionality of the system. And, although the claims are written towards the base case of one client system talking to a single server which has one database maintained on the server, there could be many servers, located centrally or spread out over a network such as the Internet, each server having one or more databases containing data for retrieval by client systems.

What is claimed is:

1. A data access system comprising:

a client computer;

a server;

a database maintained on the server, the database having

i) information to be distributed to the client computer, and

ii) a token associated with the client, wherein the token represents the status of communications between the client computer and the server;

a communication path between the client computer and the server, each client computer programmed with a specialized computer software program providing an at least one data-entry field to allow a particular user of the client computer to complete a request of information to be retrieved from the server; and

a communication path between the server and the client, allowing the server to communicate a response to the particular user's request.

2. A system as described in claim 1, wherein the status of communications between the client computer and the server includes the identity of the client and the context of any communications between the client computer and the server.

3. A system as described in claim 1, wherein the communication paths between the client and the server are stateless.

4. A system according to claim 1, wherein the communication paths between the client and the server are based upon using a Telnet communication protocol.

5. A system as described in claim 1, wherein the communication paths between the client and the server are based upon using word wide web access software.

6. A system as described in claim 5, wherein communicating with the database further includes:

a processor for exchanging an at least one plain-text message so as to establish a secure communication path through use of a public-key cryptosystem; and

an encryption processor for exchanging encrypted information between the server and the client computer after establishing the secure communication path.

7. A system as described in claim 1, wherein the communication paths between the client and the server are based upon using software which implements the Hypertext Markup Language specification.

8. A system as described in claim 1, wherein the communication paths between the client and the server are based upon using software which implements a platform-independent multi-threaded, dynamic general-purpose programming environment for dynamically creating applets and applications for accessing the database over a distributed network.

9. A system as described in claim 8, wherein the general-purpose programming environment is object-oriented.

10. A data access system comprising:

a client system and a server system in communication over a network, the client system providing requests over the network to the server system, and the server system providing data to the client system in response to the requests;

a first memory area, within the server system, for storing client context data regarding data requests from the client system;

a token, generated by the server, which identifies the first memory area, and which is returned to the client system;

a second memory area, within the client system, which stores the token received from the server system; and

an at least one request by the client which includes the token generated by the server.

11. A data access system according to claim 10, further comprising:

the at least one request including a keep_alive indicator value along with the request to the server system, so that the server system maintains an open network connection to the client system until the client system sends a countermanding request to the server system to drop the network connection.

12. An on-line transaction processing system comprising:

a database server, an application server, and a client system;

a first communication path between the database server and the application server, the first communication path being over a first network connection using a first communication protocol;

a second communication path between the application server and the client server, the second communication path being over a second network connection using a second communication protocol;

a keep_alive request transmitted by the client server to the application server in accordance with the second communication protocol, in response to which the application server reserves an area of volatile memory in response to said keep_alive request, creates a unique token identifying the area of volatile memory, and transmits said unique token to said client server in accordance with said second communication protocol; and

wherein the client server retains the unique token, and transmits the unique token to the application server in accordance with the second protocol in combination with requests for database services.

13. A method for performing an on-line transaction processing system comprising:

providing a client and a server communicating over a network using a first network communication protocol;

sending a Universal Resource Locator request to a server;

verifying that a context exists at the server for the client, and requesting information regarding the client if no such context exists;

creating a token representing the current state of communication with the client, wherein such token is transmitted to and retained by the client;

displaying a menu of information or services the client may request of the server;

selecting the information or service from the menu and transmitting a request, along with the token, to the server;

parsing the request received from the client; and if a valid request, acting upon the request; and

communicating the result to the client.

14. A method as described in claim 12, wherein the first network communication protocol is a stateless protocol, and wherein use of the token results in a second network communication protocol containing state information.

41394

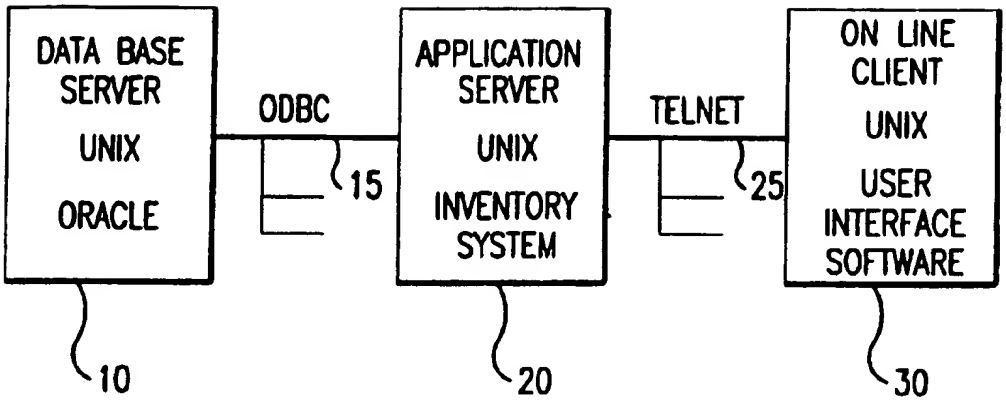


FIG.1

PRIOR ART

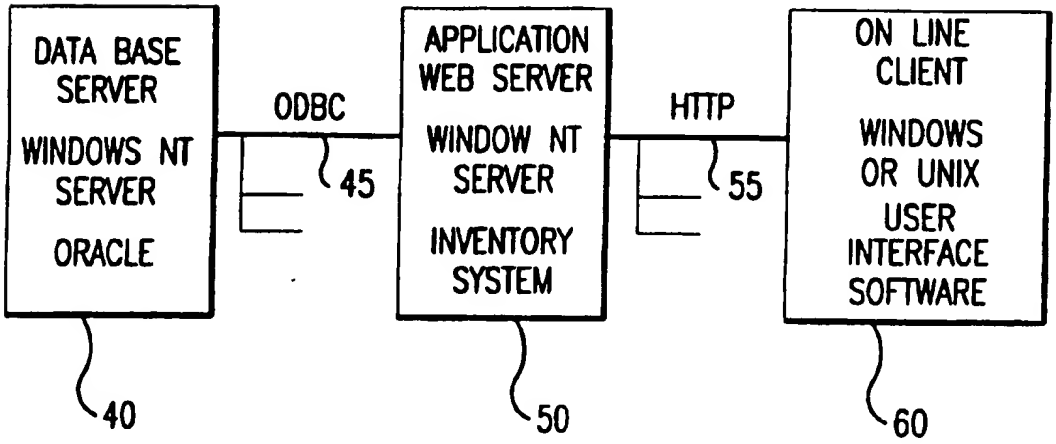


FIG.2

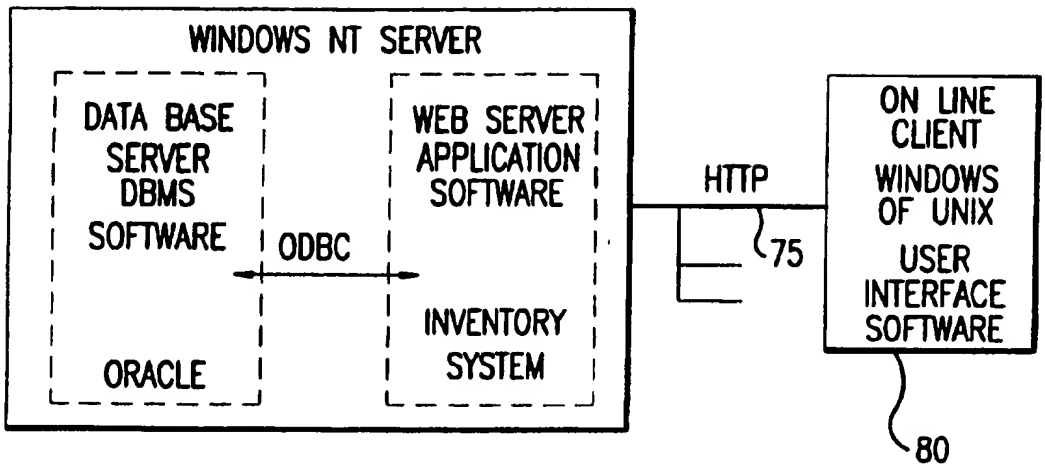

WINDOWS NT SERVER
DATA BASE SERVER DBMS SOFTWARE
ODBC
ORACLE
WEB SERVER APPLICATION SOFTWARE
INVENTORY SYSTEM
HTTP
75
ON LINE CLIENT
WINDOWS OF UNIX
USER INTERFACE SOFTWARE
80

FIG.3

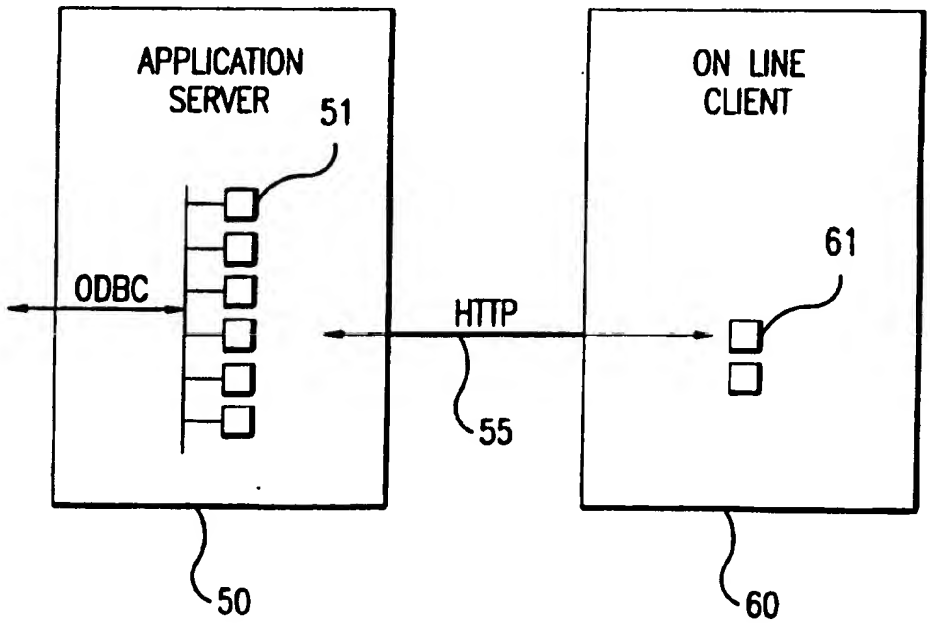

APPLICATION SERVER
51
ODBC
50
HTTP
55
ON LINE CLIENT
61
60

FIG.4

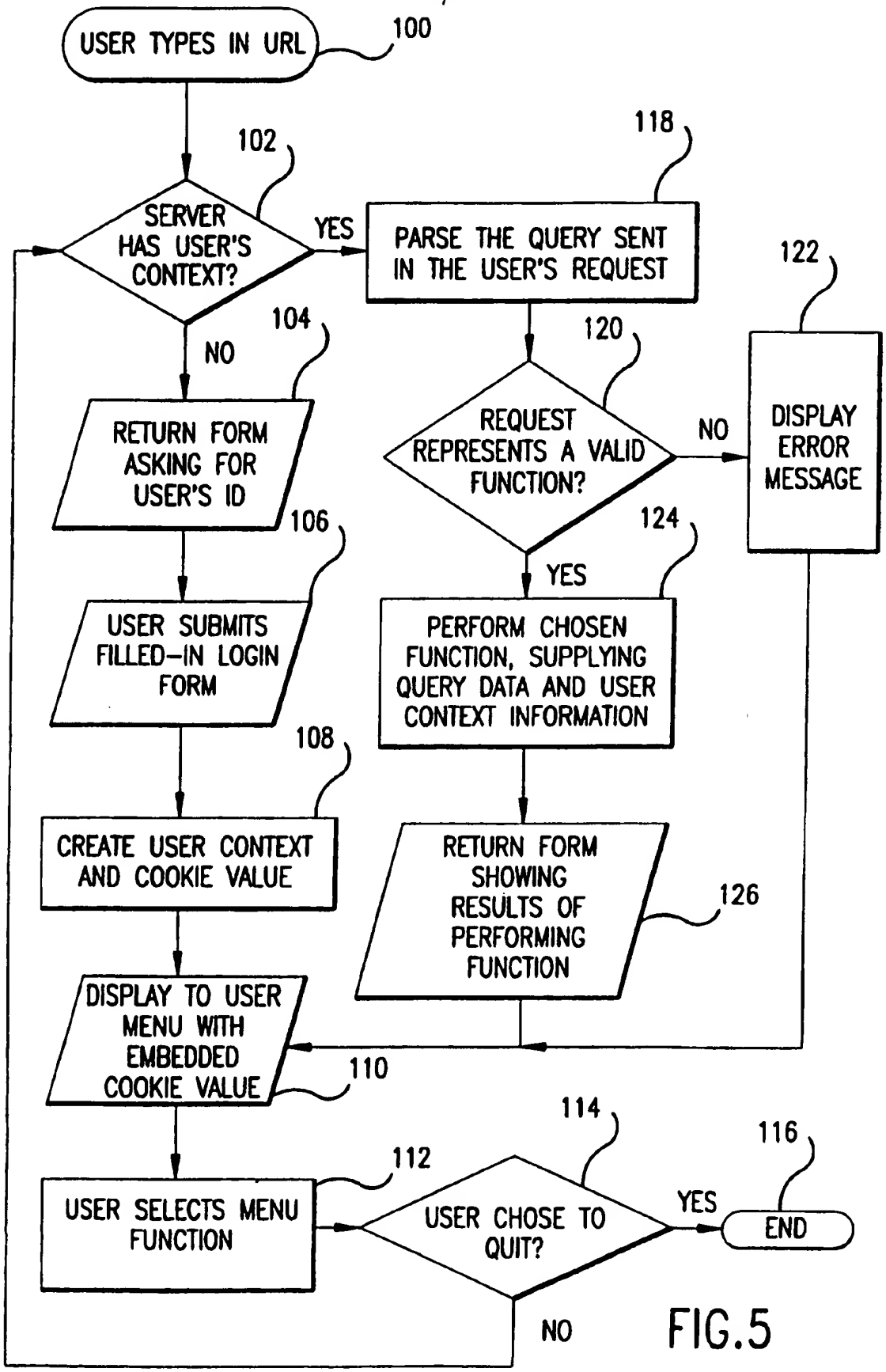


FIG.5

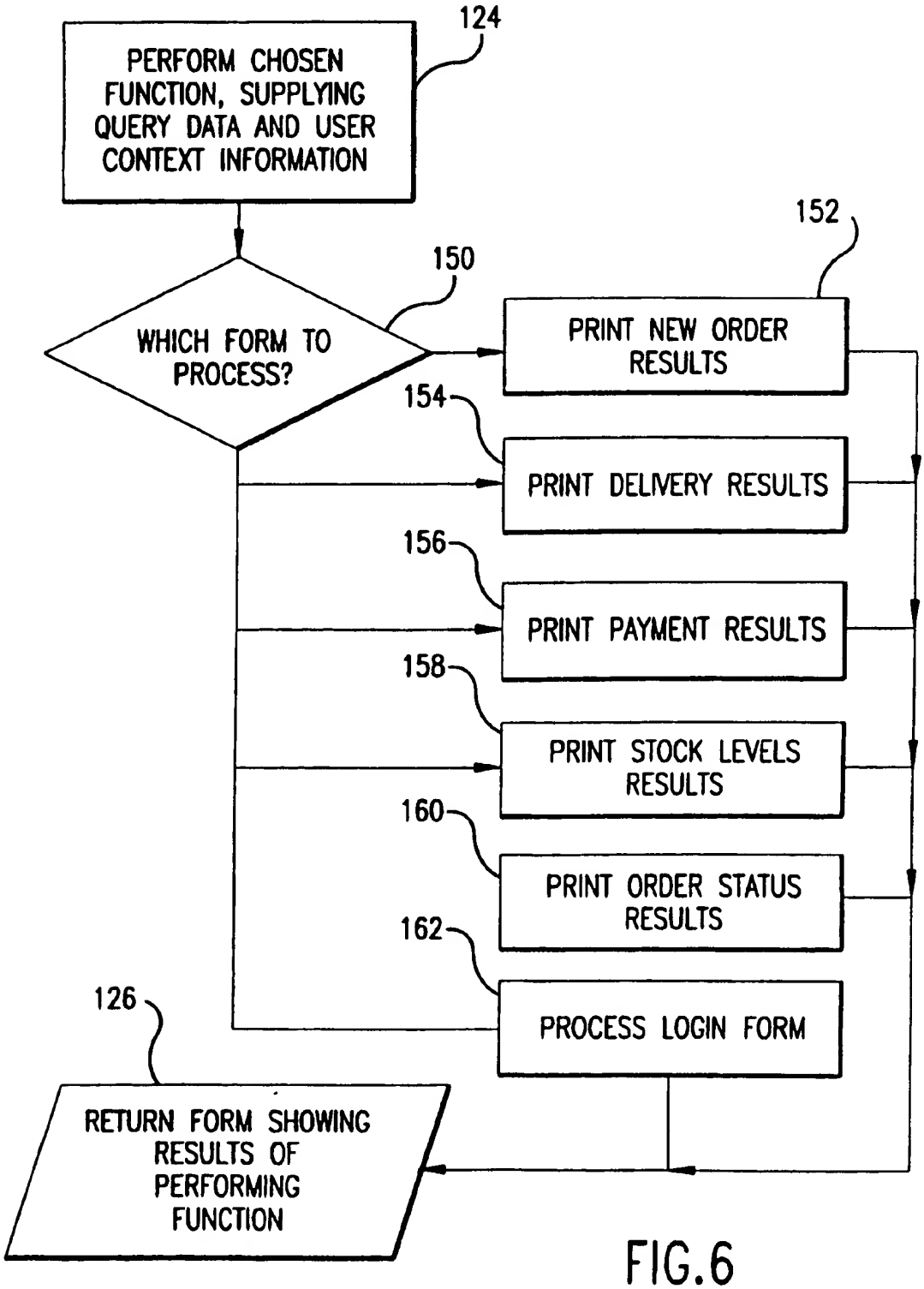

124
PERFORM CHOSEN FUNCTION, SUPPLYING QUERY DATA AND USER CONTEXT INFORMATION
150
WHICH FORM TO PROCESS?
152
PRINT NEW ORDER RESULTS
154
PRINT DELIVERY RESULTS
156
PRINT PAYMENT RESULTS
158
PRINT STOCK LEVELS RESULTS
160
PRINT ORDER STATUS RESULTS
162
PROCESS LOGIN FORM
126
RETURN FORM SHOWING RESULTS OF PERFORMING FUNCTION

FIG.6

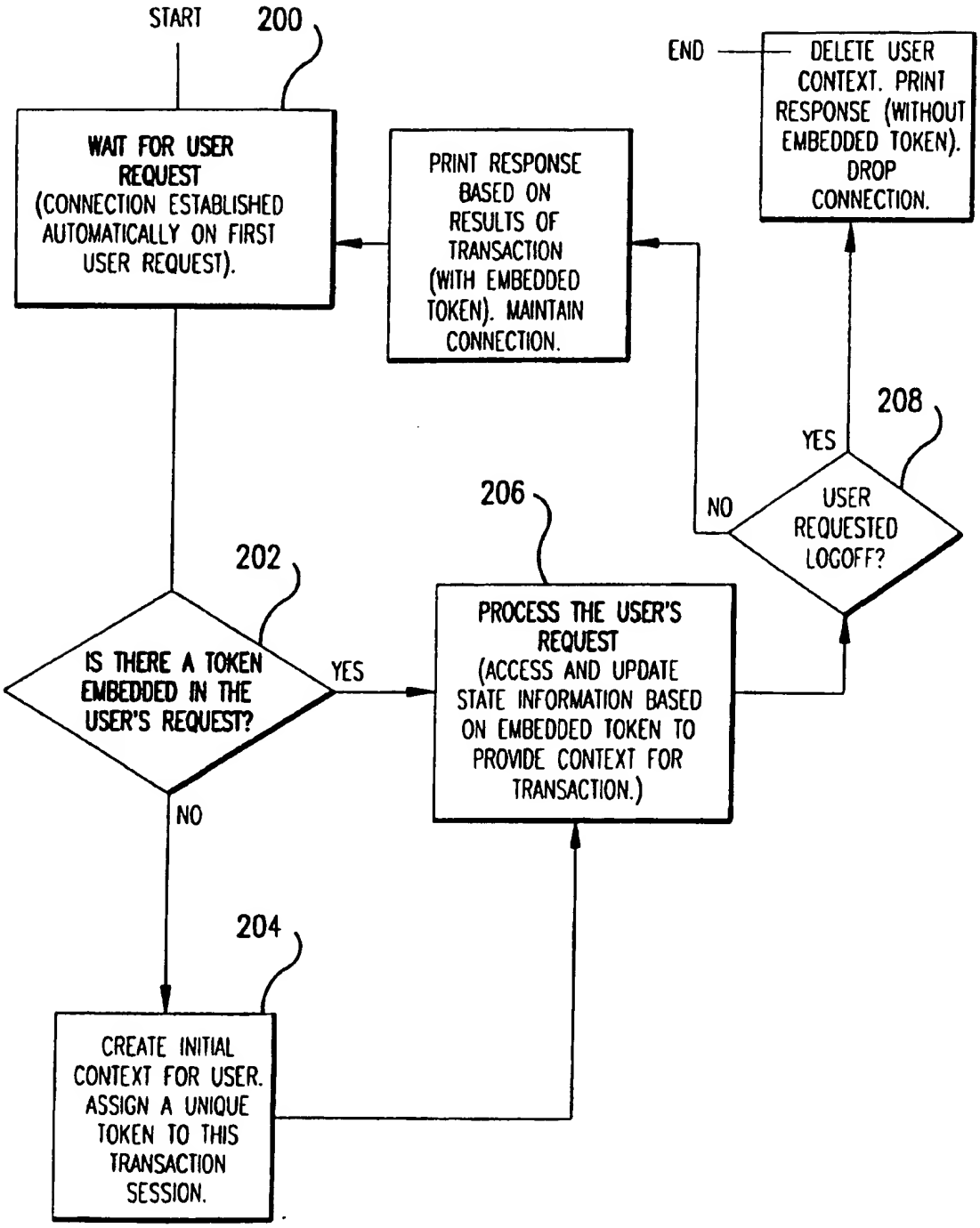


FIG.7